BÀI TẬP LÝ THUYẾT DỮ LIỆU LỚN

BÀI TẬP MÔ HÌNH MAPREDUCE

Thông tin:
Tên sv: Cao Trần Dũng
Mssv: 22410070

1. Cho dữ liệu là những tập tin nhật ký duyệt web có cấu trúc như sau:

1. Cho dữ liệu là những tập tin nhật ký duyệt web có cấu trúc như sau:
<đường dẫn>, <thời gian tính bằng phút>
Ví dụ:
https://facebook.com, 86
https://youtube.com, 33
https://tuoitre.vn, 25
https://tinhte.vn, 22
a. Thiết kế các hàm trên mô hình MapReduce thống kê thời lượng duyệt web của người dùng theo từng trang.

Bài làm:

```
// Mapper Class
class WebLogMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
  private final static IntWritable duration = new IntWritable();
  private Text url = new Text();

  public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {
    String[] fields = value.toString().split(","); // Phân tách dữ liệu theo dấu
phẩy
    if (fields.length == 2) {
      String path = fields[0].trim(); // Lấy đường dẫn
      int time = Integer.parseInt(fields[1].trim()); // Lấy thời lượng duyệt web

      url.set(path);
      duration.set(time);

      context.write(url, duration);
    }
  }
}

// Reducer Class
class WebLogReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
  private IntWritable result = new IntWritable();

  public void reduce(Text key, Iterable<IntWritable> values, Context context)
throws IOException, InterruptedException {
    int sum = 0;
    for (IntWritable val : values) {
      sum += val.get();
    }
    result.set(sum);
    context.write(key, result);
  }
}
```

b. Sử dụng 02 kỹ thuật là tổng hợp cục bộ (Combiner) trong lớp Mapper và duy trì

trạng thái biến nhớ trên toàn bộ các tác vụ map, để tối ưu hóa lớp Mapper.

Bài làm:

```
// Mapper Class with Combiner
class WebLogMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
  private final static IntWritable duration = new IntWritable();
  private Text url = new Text();

  private Map<String, Integer> durationMap; // Biến nhớ để lưu trữ thời lượng duyệt
web của từng trang

  protected void setup(Context context) throws IOException, InterruptedException {
    durationMap = new HashMap<>();
  }

  public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {
    String[] fields = value.toString().split(",");
    if (fields.length == 2) {
      String path = fields[0].trim();
      int time = Integer.parseInt(fields[1].trim());

      if (durationMap.containsKey(path)) {
        durationMap.put(path, durationMap.get(path) + time); // Tổng hợp cục bộ
thời lượng duyệt web của từng trang
      } else {
        durationMap.put(path, time);
      }
    }
  }

  protected void cleanup(Context context) throws IOException, InterruptedException
{
    for (Map.Entry<String, Integer> entry : durationMap.entrySet()) {
      url.set(entry.getKey());
      duration.set(entry.getValue());
      context.write(url, duration);
    }
  }
}
```

2. Cho dữ liệu bán hàng của một siêu thị dưới dạng tập tin có cấu trúc như sau: <mã_khách_hàng>, <mã_sản_phẩm>, <đơn_giá>, <số_lượng>

Ví dụ:

```
KH11321, SP110, 136000, 9

KH11321,    SP092,      18500,      24
KH09231,    SP003,      20300,      7
KH01287,    SP206,      94300,      10
```

a. Thiết kế các hàm trên mô hình MapReduce tính doanh số bán hàng của siêu thị đó.

Bài làm:

```
// Mapper Class
class SalesMapper extends Mapper<LongWritable, Text, Text, DoubleWritable> {
  private final static DoubleWritable sales = new DoubleWritable();
  private Text customer = new Text();

  public void map(LongWritable key, Text value, Context context) throws
```

```
IOException, InterruptedException {
    String[] fields = value.toString().split(",");

    if (fields.length == 4) {
      String customerID = fields[0].trim();
      double price = Double.parseDouble(fields[2].trim());
      int quantity = Integer.parseInt(fields[3].trim());

      double totalSales = price * quantity;

      customer.set(customerID);
      sales.set(totalSales);

      context.write(customer, sales);
    }
  }
}

// Reducer Class
class SalesReducer extends Reducer<Text, DoubleWritable, Text, DoubleWritable> {
  private DoubleWritable totalSales = new DoubleWritable();

  public void reduce(Text key, Iterable<DoubleWritable> values, Context context)
throws IOException, InterruptedException {
    double sum = 0;
    for (DoubleWritable val : values) {
      sum += val.get();
    }
    totalSales.set(sum);
    context.write(key, totalSales);
  }
}
```

b. Sử dụng 02 kỹ thuật là tổng hợp cục bộ (Combiner) trong lớp Mapper và duy trì trạng thái biến nhớ trên toàn bộ các tác vụ map, để tối ưu hóa lớp Mapper.

Bài làm:

```
// Mapper Class with Combiner
class SalesMapper extends Mapper<LongWritable, Text, Text, DoubleWritable> {
  private final static DoubleWritable sales = new DoubleWritable();
  private Text customer = new Text();

  private Map<String, Double> salesMap; // Biến nhớ để lưu trữ tổng doanh số của
từng khách hàng

  protected void setup(Context context) throws IOException, InterruptedException {
    salesMap = new HashMap<>();
  }

  public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {
    String[] fields = value.toString().split(",");
    if (fields.length == 4) {
      String customerID = fields[0].trim();
      double price = Double.parseDouble(fields[2].trim());
      int quantity = Integer.parseInt(fields[3].trim());

      double totalSales = price * quantity;
```

```
      if (salesMap.containsKey(customerID)) {
        salesMap.put(customerID, salesMap.get(customerID) + totalSales); // Tổng
hợp cục bộ doanh số của từng khách hàng
      } else {
        salesMap.put(customerID, totalSales);
      }
    }
  }

  protected void cleanup(Context context) throws IOException, InterruptedException
{
    for (Map.Entry<String, Double> entry : salesMap.entrySet()) {
      customer.set(entry.getKey());
      sales.set(entry.getValue());
      context.write(customer, sales);
    }
  }
}
```

3. Cho dữ liệu đo đạc của các cảm biến không khí dưới dạng tập tin có cấu trúc như sau:
<mã_cảm_biến>, <chỉ_số>, <độ_lớn>
Ví dụ:
1000120, SO2, 3.42
1000120, CO, 2602

1000120, PM10, 88.33
1000124, CO, 1358

Thiết kế các hàm trên mô hình MapReduce tính trung bình nồng độ khí CO của các cảm biến đo đạc được.

Bài làm:

```
// Mapper Class
class COConcentrationMapper extends Mapper<LongWritable, Text, Text,
DoubleWritable> {
  private final static DoubleWritable concentration = new DoubleWritable();
  private Text sensor = new Text();

  public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {
    String[] fields = value.toString().split(",");

    if (fields.length == 3) {
      String sensorID = fields[0].trim();
      String pollutant = fields[1].trim();
      double value = Double.parseDouble(fields[2].trim());

      if (pollutant.equalsIgnoreCase("CO")) {
        sensor.set(sensorID);
        concentration.set(value);
        context.write(sensor, concentration);
      }
    }
  }
}

// Reducer Class
class COConcentrationReducer extends Reducer<Text, DoubleWritable, Text,
```

```
DoubleWritable> {
  private DoubleWritable averageConcentration = new DoubleWritable();

  public void reduce(Text key, Iterable<DoubleWritable> values, Context context)
throws IOException, InterruptedException {
    double sum = 0;
    int count = 0;
    for (DoubleWritable val : values) {
      sum += val.get();
      count++;
    }
    double average = sum / count;
    averageConcentration.set(average);
    context.write(key, averageConcentration);
  }
}
```

4. Cho một dãy số chứa trong tập tin như sau:

```
18229
2902
45
3903
9539
33893
5
350
```

Thiết kế các hàm trên mô hình MapReduce tính trung bình nhân của dãy số trên.

Bài làm:

```
function mapper(key, value) {
  // Chuyển đổi giá trị từ chuỗi sang số nguyên
  var number = parseInt(value);

  // Phát ra cặp key-value với key là "sum" và value là số đã chuyển đổi
  emit("sum", number);

  // Phát ra cặp key-value với key là "count" và value là 1
  emit("count", 1);
}

function reducer(key, values) {
  // Kiểm tra key để xác định loại giá trị
  if (key === "sum") {
    // Tính tổng các số
    var sum = values.reduce(function(acc, curr) {
      return acc + curr;
    }, 0);

    // Phát ra cặp key-value với key là "sum" và value là tổng các số
    emit("sum", sum);
  } else if (key === "count") {
    // Đếm số lượng số
    var count = values.reduce(function(acc, curr) {
      return acc + curr;
    }, 0);
```

```
    // Phát ra cặp key-value với key là "count" và value là số lượng số
    emit("count", count);
  }
}

function finalizer(key, value) {
  // Kiểm tra key để xác định loại giá trị
  if (key === "sum") {
    // Lấy tổng các số
    var sum = value;

    // Lấy số lượng số từ giá trị cuối cùng của key "count"
    var count = getValue("count");

    // Tính trung bình nhân
    var average = sum / count;

    // Phát ra cặp key-value với key là "average" và value là trung bình nhân
    emit("average", average);
  }
}
```

5. Cho dữ liệu ghi nhận lượng điện năng tiêu thụ của từng hộ gia đình trong một tháng
dưới dạng tập tin có cấu trúc như sau:
<mã_hộ_gia_đình>, <số_điện_tiêu_thụ>
Ví dụ:

```
1000120,    325
3020180,    121
0277529,    78
0983910,    502
```

Thiết kế các hàm trên mô hình MapReduce tính tổng số tiền thu được trong tháng, biết các mức giá điện được tính theo công thức sau:
- Bậc 1: Cho kWh từ 0 – 100 là 1.734 đ
- Bậc 2: Cho kWh từ 101 – 200 là 2.014 đ
- Bậc 3: Cho kWh từ 201 – 300 là 2.536 đ
- Bậc 4: Cho kWh từ 301 trở lên là 2.834 đ

Bài làm:

```
function mapper(key, value) {
  // Tách dữ liệu thành mã hộ gia đình và số điện tiêu thụ
  var data = value.split(", ");
  var household = data[0];
  var consumption = parseInt(data[1]);

  // Tính số tiền thu được dựa trên mức giá điện
  var price;
  if (consumption <= 100) {
    price = consumption * 1.734;
  } else if (consumption <= 200) {
    price = (100 * 1.734) + ((consumption - 100) * 2.014);
  } else if (consumption <= 300) {
    price = (100 * 1.734) + (100 * 2.014) + ((consumption - 200) * 2.536);
  } else {
    price = (100 * 1.734) + (100 * 2.014) + (100 * 2.536) + ((consumption - 300) *
2.834);
  }
```

```
  // Phát ra cặp key-value với key là "total" và value là số tiền thu được
  emit("total", price);
}

function reducer(key, values) {
  // Tính tổng số tiền thu được từ các giá trị
  var total = values.reduce(function(acc, curr) {
    return acc + curr;
  }, 0);

  // Phát ra cặp key-value với key là "total" và value là tổng số tiền thu được
  emit("total", total);
}

function finalizer(key, value) {
  // Lấy tổng số tiền thu được từ giá trị cuối cùng của key "total"
  var total = value;

  // Phát ra cặp key-value với key là "total" và value là tổng số tiền thu được đã
được làm tròn đến 2 chữ số thập phân
  emit("total", total.toFixed(2));
}
```

6. Từ kết quả của bài tập 1, thiết kế các hàm trên mô hình MapReduce để tìm ra website
người dùng dành nhiều/ít thời gian xem nhất.

Bài làm:

```
// Mapper Class
class WebsiteTimeMapper extends Mapper<LongWritable, Text, Text, LongWritable> {
  private final static LongWritable time = new LongWritable();
  private Text website = new Text();

  public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {
    String[] fields = value.toString().split(",");

    if (fields.length == 3) {
      String websiteID = fields[0].trim();
      long duration = Long.parseLong(fields[2].trim());

      website.set(websiteID);
      time.set(duration);
      context.write(website, time);
    }
  }
}

// Reducer Class
class WebsiteTimeReducer extends Reducer<Text, LongWritable, Text, LongWritable> {
  private LongWritable totalDuration = new LongWritable();

  public void reduce(Text key, Iterable<LongWritable> values, Context context)
throws IOException, InterruptedException {
    long sum = 0;
    for (LongWritable val : values) {
      sum += val.get();
```

```
    }
    totalDuration.set(sum);
    context.write(key, totalDuration);
  }
}
```

7. Cho dữ liệu thông tin nhân viên dưới dạng tập tin có cấu trúc như sau: <họ>, <tên>, <chức_vụ>, <phòng_ban>, <lương>
Ví dụ:
Nguyễn Hoàng, Anh, Quản lý, Hành chính, 18650000 Trương Thị Anh, Thư, Nhân viên, Nghiên cứu, 12600000 Võ Nguyễn Thùy, Trang, Nhân viên, Nhân sự, 11680000 Trần Thanh, Nhân, Quản lý, Marketing, 20070000
Lý Thị Ngọc, Diễm, Nhân viên, Marketing, 13970000

Thiết kế các hàm trên mô hình MapReduce tính số lượng người, người có lương cao nhất/thấp nhất của từng phòng ban.

Bài làm:

```
// Mapper Class
class EmployeeMapper extends Mapper<LongWritable, Text, Text, EmployeeWritable> {
  private Text department = new Text();
  private EmployeeWritable employee = new EmployeeWritable();

  public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {
    String[] fields = value.toString().split(",");

    if (fields.length == 5) {
      String lastName = fields[0].trim();
      String firstName = fields[1].trim();
      String position = fields[2].trim();
      String departmentName = fields[3].trim();
      double salary = Double.parseDouble(fields[4].trim());

      department.set(departmentName);
      employee.set(lastName, firstName, position, salary);
      context.write(department, employee);
    }
  }
}

// Reducer Class
class EmployeeReducer extends Reducer<Text, EmployeeWritable, Text,
EmployeeWritable> {
  private EmployeeWritable highestSalaryEmployee = null;
  private EmployeeWritable lowestSalaryEmployee = null;

  public void reduce(Text key, Iterable<EmployeeWritable> values, Context context)
throws IOException, InterruptedException {
    int count = 0;
    double maxSalary = Double.MIN_VALUE;
    double minSalary = Double.MAX_VALUE;

    for (EmployeeWritable value : values) {
      count++;
      double salary = value.getSalary();

      if (salary > maxSalary) {
```

```
        maxSalary = salary;
        highestSalaryEmployee = new EmployeeWritable(value);
      }

      if (salary < minSalary) {
        minSalary = salary;
        lowestSalaryEmployee = new EmployeeWritable(value);
      }
    }

    context.write(key, new EmployeeWritable(count, null, null, maxSalary));
    context.write(key, new EmployeeWritable(count, null, null, minSalary));
    context.write(key, highestSalaryEmployee);
    context.write(key, lowestSalaryEmployee);
  }
}
```

8. Cho dữ liệu mô tả về hình đa giác (tam giác, tứ giác, ngũ giác, lục giác …) gồm các cạnh,
được lưu trữ trong tập tin như sau:

```
A--B; B--C; C--A
G--H; H--I; I--K; K--G
X--Y; Y--Z; Z--X
M--N; N--O; O--P; P--Q; Q--M
```

Cho mã giả các lớp Mapper và Reducer của chương trình thống kê số cạnh của đa giác có trong tập tin, như sau:

```
class MAPPER
method MAP(polygonId i, polygon p)
for all edge e ∈ polygon p do
EMIT(edge e, count 1)
class REDUCER
method REDUCE(edge e, counts[c1, c2,…])
sum ← 0
for all count c ∈ counts[c1, c2,…] do

sum ← sum + c EMIT(edge e, sum)
```

Sử dụng 02 kỹ thuật là tổng hợp cục bộ (Combiner) trong lớp Mapper và duy trì trạng thái biến nhớ trên toàn bộ các tác vụ map, để tối ưu hóa lớp Mapper.

Bài làm:

```
// Mapper Function
function map(polygonId, polygon) {
  let counts = {};

  for (let i = 0; i < polygon.length; i++) {
    let edge = polygon[i];
    emit(edge, 1);
  }

  function emit(edge, count) {
    if (counts[edge]) {
      counts[edge] += count;
    } else {
      counts[edge] = count;
    }
```

```
  }

  for (let edge in counts) {
    emit(edge, counts[edge]);
  }
}

// Combiner Function
function combine(edge, counts) {
  let sum = 0;
  counts.forEach(count => {
    sum += count;
  });
  return [edge, sum];
}

// Reducer Function
function reduce(edge, counts) {
  let sum = 0;
  counts.forEach(count => {
    sum += count;
  });
  emit(edge, sum);
}

// Initialize memory variable to store state across map tasks
let memory = {};

function emit(edge, count) {
  if (memory[edge]) {
    memory[edge] += count;
  } else {
    memory[edge] = count;
  }
}

// Example usage
let polygons = [
  ["A--B", "B--C", "C--A"],
  ["G--H", "H--I", "I--K", "K--G"],
  ["X--Y", "Y--Z", "Z--X"],
  ["M--N", "N--O", "O--P", "P--Q", "Q--M"]
];

polygons.forEach(polygon => {
  map(null, polygon);
});

let output = [];

for (let edge in memory) {
  output.push(reduce(edge, [memory[edge]]));
}

console.log(output);
```